SMSGate User's Guide

Contents

1. Overview

SMSGate is a bridge between LAN and the GSM network. It can be used by system administrators to monitor and control one or more computers on the LAN. What is by far its most important feature is the ability of handling files and scripts via the Raspberry PI file system, becoming a framework for future development. Its main core can be controlled via 'commands' to run user written scripts or executable files located in a specified path. The user can also get information about the system via 'orders'.

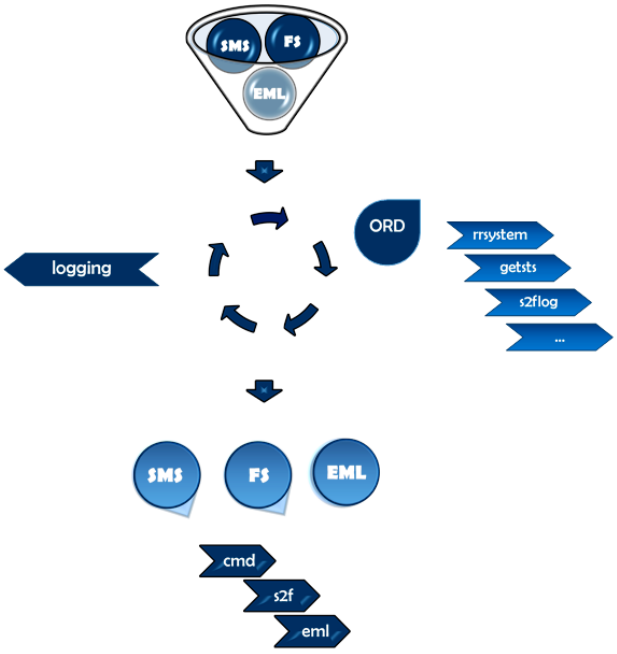


Figure 1 – SMSGate system flow diagram

The SMSGate application has the following features:

- Receive an SMS and send it via email to one or more recipients
- Receive an email and send it via SMS to one or more recipients
- Receive a command file through the file system and execute one or more commands
- Feature a phone book of employees and groups
- Be power failure safe and run even when the power is down
- Be "light" and cheap, easy to use and start up
- Implemented using Python
- Allow for future software and hardware extensions
- On-board rechargeable battery and charging circuit
- Real-time battery level monitor
- Real-time notification about important events: battery level low, Ethernet down (notification via SMS) or SMS down (notification via Ethernet)

The input data is received from three sources: GSM connection, Email and File System. An event received from one of the sources will determine an output on one of the other two data sources. All activity is logged in a text file for later review by the admin.

2. Hardware system description

The whole system is built around a Raspberry Pi 2 running a Raspbian Lite OS. The single-board computer is powered by a +5V voltage source from the Power module. Inside the Power module there are two voltage regulators providing the +5V and the +4V voltage needed for the GSM module (Quectel M95).

When the power line of +12V in down (i.e. unlike situation of a power loss) the switching logic implemented with two Schottky diodes will move almost instantaneous the input of the voltage regulators from the +12V input to the 8.4V LiPol battery providing the system with power failure immunity. The SMSGate can have an up time of up to 6 hours on battery.

The current drain from and to the battery is counted by the Coulomb counter, a small integrated circuit which gives the system an INT signal every time an amount of 0.34 mA has passed from the battery to the rest of the system or vice versa. The direction of this flow is given by the state of the POL pin, i.e. 'a low state of POL indicates the current flowing out of the battery while high impedance means the current is going into the battery' [1].

The Battery charger block manages charging of the 8.4V LiPol battery. The charging in progress is signaled on the front panel of the SMSGate by an orange led. The presence of power is signaled by a green led.

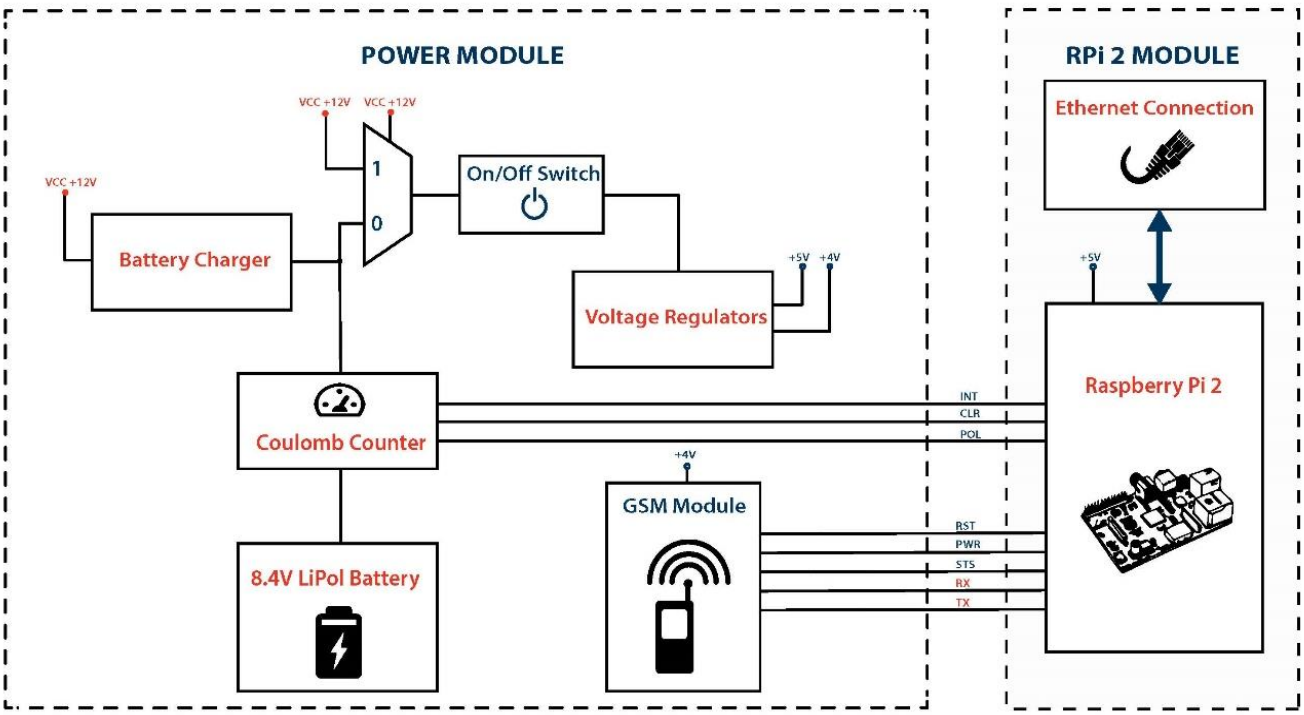


Figure 2 – Block diagram of the SMSGate system

Action / Flow	sms	eml	S2f	cmd	ord					
					rrsys	rrdmn	getsts	getlog	st2sts	s2flog
SMS	✗	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓*	✗	✓✓	✓✓
Email	✓✓	✗	✓✓	✓✓	✓✓	✓✓	✓✓*	✓✓	✓✓	✓✓
FS	✓✓**	✓✓**	✗	✗	✗	✗	✗	✗	✗	✗

* you will get a short status of your smsgate system via the sms/email to the same number/email address of the sender

** only from/to authorized members (from the phonebook)

3. Software description

3.1 General flow

The table above describes the channel switching flow for the smsgate system. A red X means that the switch is not permitted and a green tick means it is available.

The actions, keywords which must be included at the beginning of each sms/email, are in blue color. An 'ord' action must be preceded by another keyword defining the specific order, words written in gray in the table. You can read more about how the action and content should be split in the 5. Use Cases section.

3.2 Sequence diagram

This section briefly describes how the smsgate processes data through one of the main three flows.

First of all, the whole application is written in python and developed as a daemon running in the background of the Raspbian system. The regular ***sudo service start|stop|restart|status*** commands are supported by the daemon. The **sudo service smsgate status** command will print the current status of the service in terminal. If the user desires to get information about the smsgate system (threads status, GSM signal level etc.), he should enter the **statusgsm** command in the console and the result will be outputted in the txt file /var/smsgate/livestatus.txt.

The application is organized in 3 threads, one for each data flow. The worker associated with the thread checks if a new file/command has arrived and handles it. Before a new polling is made, a certain amount of time in seconds elapses as a delay. This delay can be configured in the *globalPara.py* file. Every thread sets up a flag while in idle mode (i.e. while

the delay time elapses), this being the only moment when it can be killed. This is implemented to prevent a thread to be killed while reading data from UART of retrieving an email from IMAP which could raise errors.

Sending the *sudo service smsgate stop* terminal command for the Raspbian OS will generate a wait state while the system waits for each thread to set the up his own kill flag. Only when this flag is set, the smsgate system shuts down the corresponding thread. However, if more than the usual *delays['<flow>'] +deltaTimeOut* elapses and the kill flag is not set up, the thread will be killed immediately. After killing the three threads the *systemQuit* method is called, which shuts down the Quectel M95 and quits from the IMAP and SMTP connections.

Almost every method implemented for this application return a logical True value if it has accomplished its task or False if not. The possible error for a method is handled by the callee method in almost every situation. The only error considered not resolvable by retry is the lack of communication with the QuectelM95. If more than *ATcmdNOFRetry* times in a row the QM95 does not send response to a command via UART the whole system will shut down to indicate a serious hardware issue..

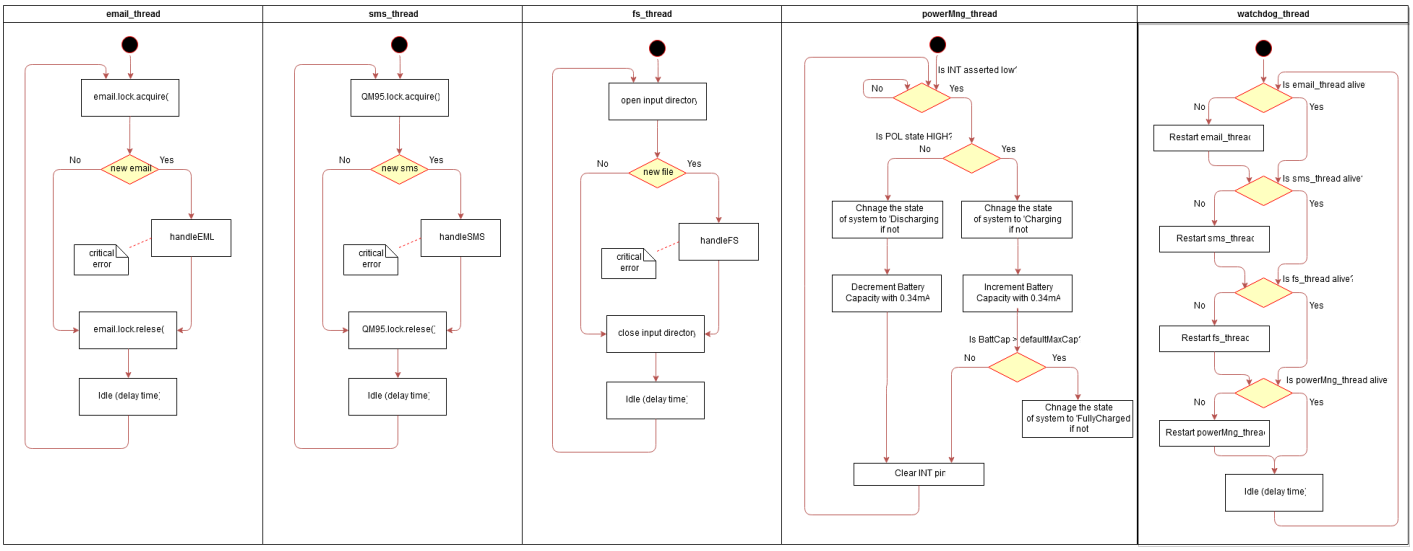


Figure 3 - Threads sequence diagram

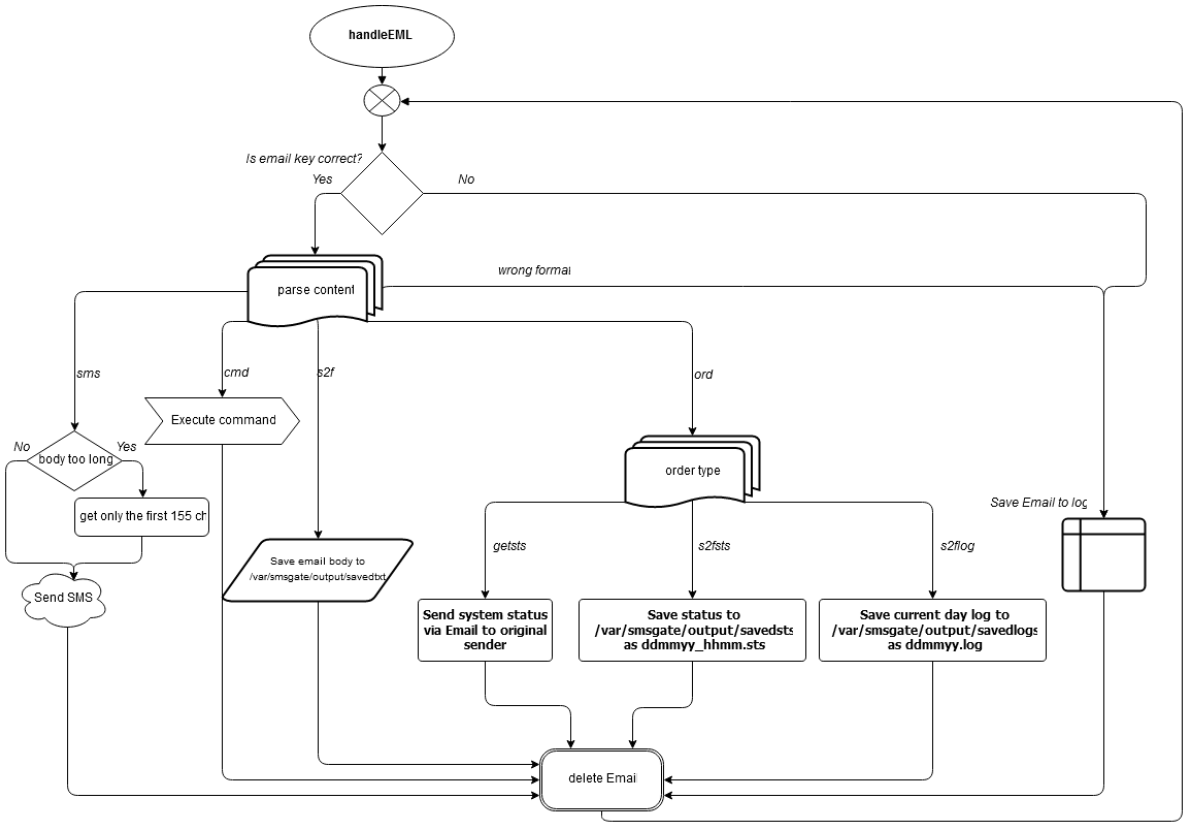


Figure 4 – Email handling method

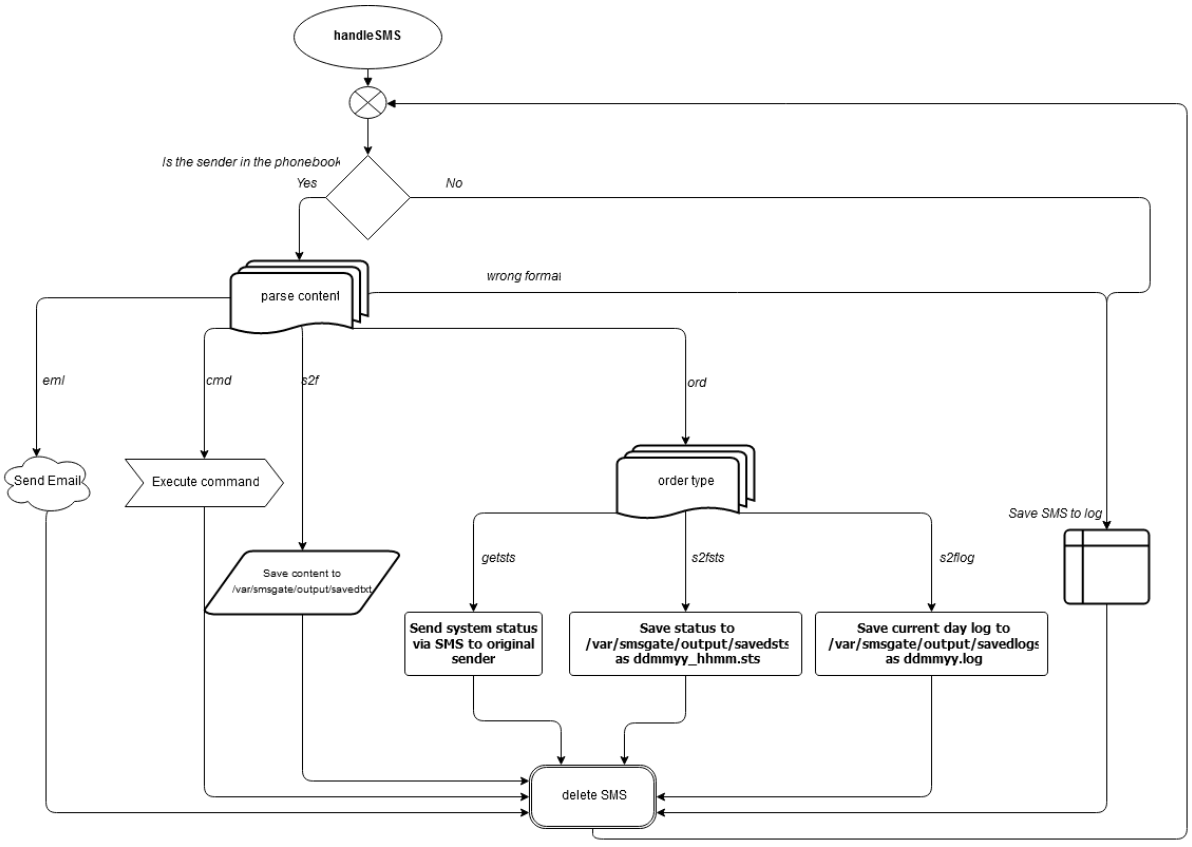


Figure 5 – SMS handling method

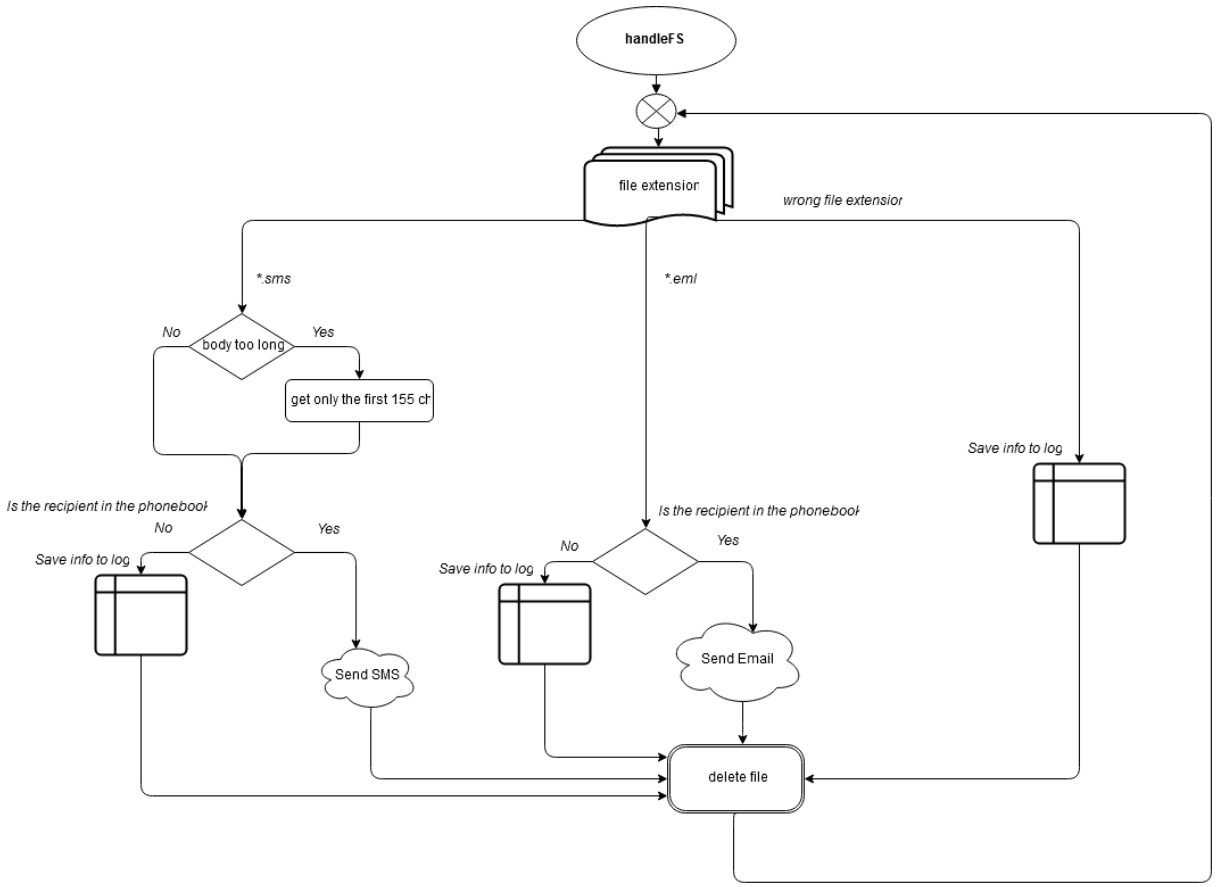


Figure 6 – File System handling method

3.3 File system

The *.py source code files are stored in the directories /home/pi/sources/py/sources and /home/pi/soruces/py/lib. In the lib directory it is located the global parameters file (*globalPara.py)* where the admin can modify some system parameter later presented in the section 3.4 Global Parameters.

Another important directory for the use of this system is /var/smsgate/, which is created by the administrator while setting up the application for the first time. More about this in section 4.1 Clone GitHub repository. The /var/smsgate contains the following:

- phonebook.json -> json file which includes all the contacts and groups of contacts relevant for the application. This is referred as the agenda of the system.

- livestatus.txt -> text file where the system outputs its status after a '*sudo service smsgate statusgsm*' command. It is called *live* because the old status is always removed from the file.
- batCap.txt -> text file used by the system to store the battery capacity. There is no need for the user to read this file since the battery capacity information is included in the *statusgsm*.
- tempalte.sms and template.eml -> template text files for a new email and sms. The user can copy them in the /var/smsgate/input directory and make a first test (after changing the phone number/email in the txt file).
- input directory -> files inserted here will be handled immediately by the system and they should followthe template described in the *5. Use cases* section.
- output directory -> this directory has 3 subdirectories: *savedlogs*, *savedsts* and *savedtxt*. Each of this subdirectory is used to save the current day log (when the 'ord#s2flog' order is received), the current moment of time gsm status (when the 'ord#s2fstsl' order is received) or a simple txt file when the action '#s2f#fileTitle#body here' is received
- logging directory -> contains the *logging.txt* log file
- commands directory -> all the executable scripts that can be executed by a *cmd* action. This folder is is mandatory because the system only searches this directory for commands.

3.4 Global parameters

The global parameters are stored in the *globalPara.py* file in /home/pi/smsgate/src/py/lib/ directory and constitute a control dashboard of the application. The administrator must modify these parameters to control the smsgate. There are several categories of parameters:

- General parameters -> general parameters for the system such as the UART speed or hardware connection pins for the Quectel M95 or the coulomb counter.
- Username and passwords -> Username and password for the email connection. To access another email account via IMAP (retrieving email) and SMTP (sending email) the administrator must change the login address and password. The email key is defined as a string mandatory at the end of the subject for a received email. More about this in the *5. Use cases* section.
- Restrictions specifications -> The system is designed to verify if the user provides the right parameters to an action. An email address is considered valid only if its domain is among the 'allowed_domains' and if its username does not contain *special_EMLchars*. For the output file name in the s2f action, the system verifies if it does not contain

special_FSchars. Finally, a phone number is considered valid only if it has exactly 10 digits in local format (e.g. 0724000000).

- Global variables -> In this section the admin can modify global variables for the system. The most important is the 'delays' dictionary, which contains the delay time between two iterations in the sms thread, email or fs thread. Another important parameter which should be mentioned is the ATcmdNOFRetry, which is the number or retries the system does if no response is received for a command sent to QuectelM95 before quitting. The other parameters are clearly explained by their suggestive name. More about the behavior of the system according to these parameters in the *3.1 General flow* section.
- Program Paths -> String parameters are defined here to locate the resources for the system. Even if it is not recommended, these paths can be changed, the admins having this option in the *global parameters* file.
- Logging -> The logging format can be customized here (e.g. date format, message format)

3.5 Phonebook

The phonebook whose path is indicated by the *phonebook_path* (global parameter) contains a list of the known people. For each person there are some mandatory attributes. These are **ID**, **Name**, **Email** and **PhoneNo**. The sysadmin must modify these parameters to fit his needs. A theoretically unlimited amount of registrations can be introduced here. The phonebook file is organized as a *json* file and adding a new registration must obey the default template. If an employee has no email or phone number, the field must be completed with an empty string.

There are also groups defined in this json file and membership to a specified group is defined by adding the employee ID in the respective group list. Note that the employee ID is introduced as an integer in this area. Groups are adding a powerful feature to *smsgate*, that of sending a notification to a whole predefined group rather than to a list of people. The group alias (e.g. admins) should be remembered by the user because will be used to define the attribute of a future action.

The json file is parsed and inserted into the system only once when the smsgate starts or restarts. If a change is made in the json file, the daemon must be restarted for this change to take effect.

```
{
  "employees": [
    {
      "ID": "1",
      "Name": "Ives Tamboon",
      "PhoneNo": "0724000000",
      "Email": "ives.tamboon@gmail.com"
    },
    {
      "ID": "2",
      "Name": "Leila Dopoller",
      "PhoneNo": "0724000000",
      "Email": "leila.dopoller@gmail.com"
    },
    {
      "ID": "3",
      "Name": "Helga Gosond",
      "PhoneNo": "0724000000",
      "Email": "helga.gosond@gmail.com"
    },
    {
      "ID": "4",
      "Name": "Emory Spesper",
      "PhoneNo": "0724000000",
      "Email": "emory.spesper@gmail.com"
    },
    {
      "ID": "5",
      "Name": "Bertha Loltent",
      "PhoneNo": "0724000000",
      "Email": "bertha.loltent@gmail.com"
    }
  ],
  "groups": {
    "admins": [
      1,
      2
    ],
    "sysadmins": [
      1,
      2,
      3
    ],
    "staff": [
      1,
      5,
      3
    ]
  }
}
```

Figure 2 – phonebook.json example file

4. Install and setup

4.1 Clone GitHub repository

First, if not installed already, you need to install Git on Raspbian:

```
sudo apt-get install git
```

Then clone the GitHub repository on your Raspbian system:

```
git clone https://github.com/razdrian/smsgate.git
```

Now create a new folder called **smsgate** in **/var/** and copy into it the entire contents of the **/var_smsgate** folder from the repository you just cloned. If /var/smsgate/input, /var/smsgate/output/savedsts, /var/smsgate/output/savedlogs and /var/smsgate/output/savedtxt directories are not created make sure to create on your own or otherwise the application will now work properly.
!!!!! Application should be installed using a dedicated unix user/group. It must not necessarily be root user. !!!!!

4.2 How to start smsgate service

Move the smsgate file???which file??? from the repository to */etc/init.d*. This file contains the application which runs as a service in Raspbian. Create a symbolic link called **K02smsgate** in directories **/etc/rc0.d /etc/rc1.d** and **/etc/rc6.d** by typing the following command each time in the corresponding directory:

```
sudo ln -s /etc/init.d/smsgate K02smsgate
```

Create a symbolic link called **S01smsgate** in directories **/etc/rc2.d /etc/rc3.d, /etc/rc4.d** and **/etc/rc5.d** by typing the following command each time in the corresponding directory:

```
sudo ln -s /etc/init.d/smsgate S01smsgate
```

Now the smsgate application will run as a service in Raspbian and you can also send it commands such as start, stop, restart, status or statusgsm. A detailed status of the system will be outputted to /var/smsgate/livestatus.txt file every time you type:

```
sudo service smsgate statusgsm
```

From this time on, the smsgate application will run as a daemon and will start every time the Raspberry Pi starts and will shut down every time the Raspberry Pi shuts down.

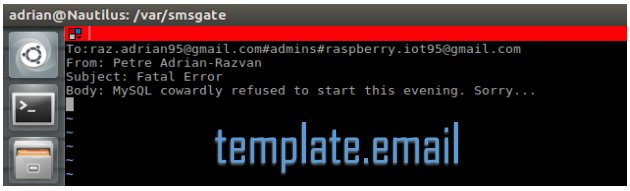
4.3 Service commands

The smsgate service supports the basic commands of any other linux daemon such as **start|stop|restart|status.** Moreover, a **statusgsm** command is available to get the smsgate system status in the /var/smsgate/livestatus.txt file since the regular **status** command will only give you the linux daemon status (up and running, up time etc).

5. Use cases

- **FS to Email**

FROM: Any RPi User with access to /var/smsgate/input/ directory
TO: Only users from phonebook
The file MUST be saved in *input_path* with the format <file>.email
e.g. tempalte.email



single recipient:
- **To:** raz.adrian95@gmail.com

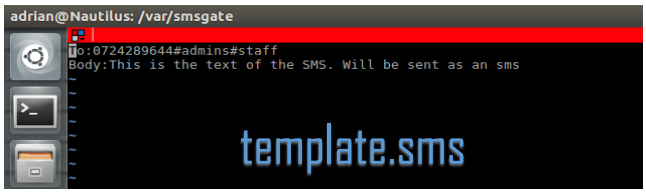
multiple recipients:
- **To:** raz.adrian95@gmail.com, raspberry.iot95@gmail.com

group recipients:
- **To:** Admin#Staff

- **Subject:** My SQL has stopped
- **Body:** Today at around 9PM the mySQL server has stopped!

- **FS to SMS**

FROM: Any RPi User with access to /var/smsgate/input directory
TO: Only users from phonebook
The file MUST be saved in *input_path* with the format <file>.sms
e.g. template.sms



single recipient:
- **To:** 0724000000

multiple recipients:
- **To**: 0724000000#0724000001

group recipients:
- **To:** Users

- **Body:** Today the apache server died but was resurrected!

- **Email to SMS**

FROM: Any Email address, but only specified subject format
TO: any valid Phone Number or Group
e.g.
single recipient:
- **Mail Subject:** sms#0724000000#6FA32E

multiple recipients:
- **Mail Subject:** sms#0724000000#0724000001#6FA32E

group recipients:
- **Mail Subject:** #Admin#6FA32E
- **Mail Body:** This is an SMS send from smsgate system!

* (no more than 160 characters. or only the first 160 will be sent)

- **Email to FS**

FROM: Any Email address, but only specified subject format
TO: Raspberry Pi File System
e.g.
- **Mail Subject:** s2f#fileTitle#6FA32E
- **Mail Body:** This is a text file located in the /var/smsgate/output/savedtxt directory

- **SMS to FS**

FROM: Only users form the phonebook
TO: Raspberry Pi File System
e.g.
- **SMS Body:** s2f#sputnik#This text will be saved in the output path directory under the name sputnik.

- **Email to cmd**

FROM: Any Email address, but only specified subject format
TO: SMSGate core
e.g.
single command:
- **Mail Subject:** cmd#blinkred.py#6FA32E
- **Body:** void

multiple command:
- **Mail Subject:** cmd#blinkred.py#blinkyellow.py#6FA32E
- **Body:** void

- **SMS to command**

FROM: Only users form the phonebook
TO: SMSGate core
e.g.
single command:
- **SMS Body:** cmd#backup_sql.sh

multiple commands:
- **SMS Body:** cmd#backup_sql.sh#restart_server.sh

- **Email to cmd**

FROM: Only users form the phonebook
TO: any valid email address or group
e.g.
single recipient:

- **SMS Body:** eml#raz.adrian95@gmail.com#Emmergency from Petre#I have no internet conn. Call me ASAP!

multiple recipients:

- **SMS Body:** eml#raz.adrian95@gmail.com#raspberry.iot95@gmail.com#Bad day#I have lost my toothbrush!

group recipients:

- **SMS Body:** eml#Admin#EMMERCENCY#The power is off!

6. References

[1] Terminator – useful tool for arranging terminals
[2] Pycharm Edu - free, easy and professional Python programming tool
[3] OrCAD Cadence Lite – OrCAD Capture CIS for schematic diagram design and PCB editor for Gerber files
[4] GerbTool by WISE Software, free trial download for inspecting Gerber files.
[5] Fusion 360 AUTODESK – 3D CAD and CAE tool
[6] About Epydoc documentation
[7] Free UART from RaspberryPi
[8] Logging in Python
[9] About IMAP and SMTP
[10] Unitest in Python
[11] Unix daemon in Python
[12] Python script running as Daemon in Linux
[13] Poplib in Python
[14] Parsing a raw email
[15] Imaplib example using Gmail
[16] Threading in Python
[17] Getting a Python script to run in the background
[18] Disable Bluetooth to make serial work on RPi3
[19] Best Guide to Li-ion and LiPo battery charger
[20] How to make your own PCBs at home

7. Licenses